**7.Basic Spring Boot Application with Spring Data JPA**

**Aim:**To create the Basic Spring Boot Application with Spring Data JPA

**Description:**

This project demonstrates how to create a basic Spring Boot application integrated with Spring Data JPA. It provides a simple setup for building Java applications that interact with relational databases using JPA and Hibernate. The application includes:

* Spring Boot for rapid application development and simplified configuration.
* Spring Data JPA for database operations such as saving, updating, deleting, and retrieving records without writing boilerplate SQL queries.
* Entity classes to represent database tables.
* Repository interfaces for CRUD operations.
* REST controllers (optional) to expose endpoints for interacting with the database.

This serves as a foundational template for beginners to understand how to set up Spring Boot with JPA and can be extended for real-world applications.

**Program:**

**Student.java**

package com.example;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

@Entity

public class Student {

@Id

private int sno;

private String sname;

public Student() {}

public Student(int sno, String sname) {

super();

this.sno = sno;

this.sname = sname;

}

public int getSno() {return sno;}

public void setSno(int sno) {this.sno = sno;}

public String getSname() {return sname;}

public void setSname(String sname) {this.sname = sname;}

}

**StudentController.java**

package com.example;

import java.util.List;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RestController;

import org.springframework.web.bind.annotation.RequestMapping;

@RestController

//@RequestMapping("/students")

public class StudentController {

private final StudentRepository repo;

public StudentController(StudentRepository repo) {

this.repo = repo;

}

//@RequestMapping("/students")

//Add new student

@PostMapping

public Student addStudent(@RequestBody Student student) {

return repo.save(student);

}

//Get all students

@GetMapping

public List<Student> getAllStudents(){

return repo.findAll();

}

}

**StudentRepository.java**

package com.example;

import org.springframework.data.jpa.repository.JpaRepository;

public interface StudentRepository extends JpaRepository<Student,Integer> {

}

**U3StudentApplication.java**

package com.example;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.Bean;

@SpringBootApplication

public class U3StudentApplication {

public static void main(String[] args) {

SpringApplication.run(U3StudentApplication.class, args);

}

@Bean

CommandLineRunner initDatabase(StudentRepository repo) {

return args ->{

repo.save(new Student(1,"Swarupa"));

repo.save(new Student(2,"Gayatri"));

repo.save(new Student(3,"Sridevi"));

System.out.println("Students inserted!");

};

}

}

**application.properties**

spring.application.name=U3Student

server.port=0000

spring.datasource.url=jdbc:mysql://localhost:3306/demo

spring.datasource.username=root

spring.datasource.password=Swarupa@123

spring.jpa.hibernate.ddl-auto=create-drop

spring.jpa.show-sql=true

**pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.5.5</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com</groupId>

<artifactId>U3StudentApplication</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>U3Student</name>

<description>Demo project for Spring Boot</description>

<url/>

<licenses>

<license/>

</licenses>

<developers>

<developer/>

</developers>

<scm>

<connection/>

<developerConnection/>

<tag/>

<url/>

</scm>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jdbc</artifactId>

</dependency>

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

<version>3.5.2</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

1. **Pagination and Sorting in Spring Data JPA**

**Aim:**To create the Pagination and Sorting in Spring Data JPA

**Description:**

Pagination and sorting are essential features when dealing with large sets of data in a database. Instead of retrieving all records at once, which can affect performance, Spring Data JPA provides built-in support to fetch data in chunks (pages) and in a specific order.

**Pagination:** It allows dividing the result set into smaller parts (pages). This helps reduce memory usage and speeds up response times. Spring Data JPA provides the Pageable interface and the PageRequest implementation to define the page number, size (number of records per page), and sorting details. The result is returned as a Page<T> or Slice<T> object.

**Sorting:** Sorting enables arranging the records in ascending or descending order based on one or more columns. Spring Data JPA supports sorting via the Sort class, which can be passed along with the Pageable object or used independently.